DUYỆT ĐỒ THỊ ƯU TIÊN CHIỀU SÂU
(Depth First Search - DFS)

I. DFS trên đa đồ thị vô hướng

```
// Input:
int n;                          // Số đỉnh đồ thị
// Danh sách đỉnh kề , first - tên đỉnh, second - tên cạnh
vector<pair<int,int> > adj[maxn];

// Output:
int num[maxn];                  // Thứ tự duyệt DFS
int low[maxn];                  // Thứ tự ngược DFS
int start[maxn];                // Vị trí bắt đầu duyệt DFS
int stop[maxn];                 // Vị trí kết thúc duyệt DFS
int depth[maxn];                // Chiều sâu của cây
int p[maxn];                    // đỉnh cha DFS
int pe[maxn];`                  // Số hiệu cạnh đến đỉnh cha (chỉ khi là đa đồ thị)

// Biến nháp
int color[maxn], pe[maxn];
int id = 0;

// Hàm chính gọi đệ qui
void DFS(int u) {
    color[u] = 1;
    num[u] = low[u] = ++id;
    start[u] = id;
    for(auto e : adj[u]) {
        int v = e.first, idx = e.second;
        if (idx != pe[u]) {
            if (color[v] == 0) {
                depth[v] = depth[u] + 1;
                p[v] = u;
                pe[v] = idx;
                dfs(v);
                low[u] = min(low[u], low[v]);
            } else
                low[u] = min(low[u], num[v]);
        }
    }
    stop[u] = id;
}

// Chương trình chính:

    for(int i = 1; i <= n; ++i)
        color[i] = 0;
    id = 0;
    for(int i = 1; i <= n; ++i)
        if (color[i] == 0)
            dfs(i);
```

II. Xác định đỉnh khớp trên đa đồ thị vô hướng

```
void dfskhop(int u) {
    color[u] = 1;
    khop[u] = 0;
    num[u] = low[u] = ++id;
    int socon = 0;
    for(auto e : adj[u]) {
        int v = e.first, idx = e.second;
        if (idx != pe[u]) {
            if (color[v] == 0) {
                ++socon;
                pe[v] = idx, p[v] = u;
                dfskhop(v);
                low[u] = min(low[u], low[v]);
                if (low[v] >= num[u])
                    ++khop[u];
            } else
                low[u] = min(low[u], num[v]);
        }
    }
    if (p[u] == 0)
        khop[u] = socon - 1;
}
```

III. Xác định cạnh cầu trên đa đồ thị vô hướng

```
Input:
int num[maxn], low[maxn];
pair<int,int> edge[maxm];

Output:
int cau[maxn];                  // cau[i]=1 nếu cạnh i là cầu, ngược lại, cau[i]=0;

Code:
for(int i=1;i<=m;++i) {
    int u=edge[i].first, v=edge[i].second;
    if (p[u]==v) swap(u,v);
    cau[i]=(p[v]==u && low[v]>num[u]);
}
```

IV. Xác định các thành phần liên thông mạnh trên đồ thị có hướng

```
Input:
int n;
vector<int> adj[maxn];

Output:
int cnt_strong_connected;                   // Số thành phần liên thông mạnh
int strong_connected[maxn];                 // Số hiệu TPLTM của mỗi đỉnh
vector<int> list_strong_connected[maxn];          // Danh sách mỗi đỉnh của TPLTM

stack<int> st;
```

```
void tarjan(int u) {
    color[u] = 1;
    num[u] = low[u] = ++id;
    st.push(u);
    for(int v : adj[u]) {
        if (color[v] == 0) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else
            if (color[v] == 1)
                low[u] = min(low[u], num[v]);
    }

    // Lay ra mot thanh phan lien thong manh
    if (low[u] == num[u]) {
        ++cnt_strong_connected;
        int v;
        do {
            v = st.top();
            st.pop();
            strong_connected[v] = cnt_strong_connected;
            list_strong_connected[cnt_strong_connected].push_back(v);
            color[v] = 2;
        } while (v != u);
    }
}
```

V. Thành phần song liên thông cạnh trên đa đồ thị vô hướng

```
Input:
int n;
vector<pair<int,int> > adj[maxn];

Output:
int cnt_biconnected;                          // Số thành phần song liên thông cạnh
int biconnected[maxn];                        // Số hiệu STL Cạnh mỗi đỉnh
vector<int> list_biconnected[maxn];           // Danh sách mỗi đỉnh của SLT


stack<int> st;
void tarjan(int u) {
    color[u] = 1;
    num[u] = low[u] = ++id;
    st.push(u);
    for(auto e : adj[u]) {
        int v = e.first, idx = e.second;
        if (idx != pe[u]) {
            if (color[v] == 0) {
                p[v] = u;
                pe[v] = idx;
                tarjan(v);
                low[u] = min(low[u], low[v]);
            } else
                low[u] = min(low[u], num[v]);
        }
```

```
    }

    // Lay ra mot thanh phan song liên thông cạnh
    if (low[u] == num[u]) {
        ++cnt_biconnected;
        int v;
        do {
            v = st.top();
            st.pop();
            biconnectd[v]=cnt_biconnected;
            list_biconnected[cnt_biconnected].push_back(v);
        } while (v != u);
    }
}
```

VI. Thành phần song liên thông đỉnh trên đa đồ thị vô hướng

```
Input:
int n;
vector<pair<int,int> > adj[maxn];

Output:
int cnt_biconnected;                          // Số thành phần song liên thông cạnh
vector<int> list_biconnected[maxn];           // Danh sách mỗi đỉnh của SLT


stack<int> st;
void tarjan(int u) {
    color[u] = 1;
    num[u] = low[u] = ++id;
    st.push(u);
    for(auto e : adj[u]) {
        int v = e.first, idx = e.second;
        if (idx != pe[u]) {
            if (color[v] == 0) {
                p[v] = u;
                pe[v] = idx;
                tarjan(v);
                low[u] = min(low[u], low[v]);
            } else
                low[u] = min(low[u], num[v]);
        }
    }

    // Lay ra mot thanh phan lien thong manh
    int w = p[u];
    if (low[u] >= num[w]) {
        ++cnt_biconnected;
        int v;
        do {
            v = st.top();
            st.pop();
            list_biconnected[cnt_biconnected].push_back(v);
        } while (v != u);
        if (low[u] == num[w] || list_biconnectd.size()==1)
```

```
            list_biconnected[cnt_biconnected].push_back(w);
    }
}
```

VII. Sắp xếp topo trên DAG

```
// Code:
int pos[maxn];
int cnt=n+1;
void topo(int u) {
    color[u]=1;
    for(int v : adj[u]) {
        if (color[v]==0) {
            topo(v);
        }
    }
    pos[--cnt]=u;
}
```

VIII. Đếm số đỉnh trong cây con DFS

```
int sz[maxn];
void DFS(int u, int dad) {
    sz[u] = 1;
    for(int v : adj[u])
        if (v != dad) {
            DFS(v, u);
            sz[u] += sz[v];
        }
}
```

IX. Tìm chu trình Euler trên đồ thị vô hướng

```
int n, m, deg[maxn];
pair<int, int> E[maxm];
vector<int> adj[maxn];

stack<int> s;
vector<int> path;
int cur[maxn];
int nho[maxm];

void Euler(int xp) {
    s.push(xp);
    while (!s.empty()) {
        int u = s.top();
        while(cur[u] < adj[u].size() && nho[adj[u][cur[u]]])
            ++cur[u];
        if (cur[u] >= adj[u].size()) {
            path.push_back(u);
            s.pop();
        } else {
            int i = adj[u][cur[u]];
```

```
            int v = (E[i].first == u) ? E[i].second : E[i].first;
            s.push(v);
            nho[i] = 1;
        }
    }
    if (path.size() < m + 1) {
        cout << "IMPOSSIBLE";
        return;
    }
    for(int x : path)
        cout << x << ' ';
}
```

X. Tìm chu trình Euler trên đồ thị có hướng

```
int n, m, deg[maxn];
pair<int, int> E[maxm];
vector<int> adj[maxn];

stack<int> s;
vector<int> path;
int cur[maxn];
int nho[maxm];

void Euler(int xp) {
    s.push(xp);
    while (!s.empty()) {
        int u = s.top();
        if (cur[u] >= adj[u].size()) {
            path.push_back(u);
            s.pop();
        } else {
            int v = adj[u][cur[u]++];
            s.push(v);
        }
    }
    reverse(path.begin(), path.end());
    if (path.size() < m + 1) {
        cout << "IMPOSSIBLE";
        return;
    }
    for(int x : path)
        cout << x << ' ';
}
```